Introspection Into Systemverilog Without Turning It Inside Out

Dave Rich
Mentor Graphics 46871 Bayside Parkway Fremont, CA 94538 dave_rich@mentor.com

Comparing VPI versus DPI code

```
typedef struct { int A,B; } AB_s;
AB_s ab = '{1,2};
int result;

initial result=$sum(ab) // should return 3
```

```
typedef struct { int A,B; } AB_s;
AB_s ab = '{1,2};
int result;
import "DPI-C" context function sum(AB_s arg);

initial result=sum(ab) // should return 3
```

VPI

```
int calltfSum(char* user_data) {
 vpiHandle systf_h, itr_h, arg_h,
 s_vpi_value argVal;
 int A, B;
  systf_h = vpi_handle(vpiSysTfCall, NULL);   /* call instance */
  itr_h = vpi_iterate(vpi_Argument, systf_h); /* iterate over arguments */
  arg_h = vpi_scan(itr_h);         /* get handle to 1st argument */
  vpi_free_object(itr_h);          /* housekeeping */
  itr_h = vpi_iterate(vpiMember,arg_h); /* iterate over struct members */
  arg_h = vpi_scan(Itr_h);         /* handle to A member */
  argVal.format = vpiIntVal;
  vpi_get_value(arg_h, &argVal); /* get A value in int format */
  A = argVal.value.integer;
  arg_h = vpi_scan(itr_h);         /* handle to B member */
  vpi_get_value(arg_h, &argVal); /* get B value in int format */
  B = argVal.value.integer;
  argVal.value.integer = A + B;
  vpi_put_value(systf_h, &argVal, NULL, vpiNoDelay); /* function return value */
  vpi_free_object(itr_h);          /* housekeeping */
}
```

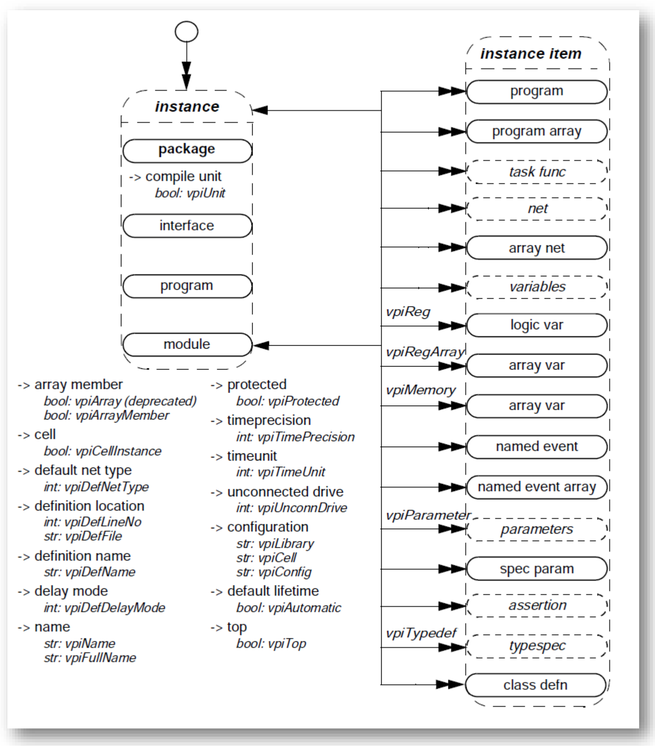
DPI

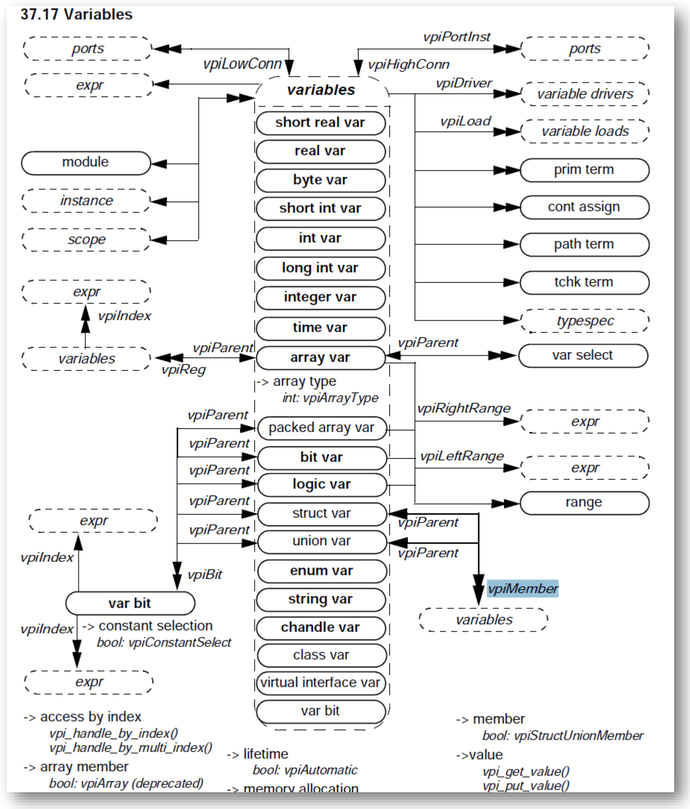
```
typedef struct { int A,B; } AB_s;

int sum( const AB_s* arg) {
  return arg->A + arg->B;
}
```

15 lines of VPI code reduced to 1 line of DPI code

VPI Object Model Diagrams





VPI through DPI Application

```
module top;
  typedef struct { int A,B; } AB_s;
  AB_s ab = '{1,2};
 vpiHandle itr_h, arg_h,
 int A, B, result;
  initial begin
    arg_h = vpi_handle_by_name("top.a");   /* get handle to ab variable */
    vpi_free_object(itr_h);          /* housekeeping */
    itr_h = vpi_iterate(vpiMember,arg_h); /* iterate over struct members */
    arg_h = vpi_scan(Itr_h);         /* handle to A member */
    A = vpi_get_int_value(arg_h);  /* get A value in int format */
    arg_h = vpi_scan(itr_h);         /* handle to B member */
    B = vpi_get_int_value(arg_h);  /* get B value in int format */
    B = argVal.value.integer;
    result = A + B;
    vpi_free_object(itr_h);          /* housekeeping */
  end
endmodule
```

VPI C code re-written as VPI SystemVerilog code

DPI Package Specification

```
//------------------------------------------------------------------------
//   Copyright 2005-2016 Mentor Graphics Corporation
//
//   Licensed under the Apache License, Version 2.0 (the"License"); you may not use this file except in
//   compliance with the License.  You may obtain a copy of the License at
//
//       http://www.apache.org/licenses/LICENSE-2.0
//
//   Unless required by applicable law or agreed to in writing, software distributed under the License is
//   distributed on an "AS IS" BASIS, WITHOUT WARRANTIES ORCONDITIONS OF ANY KIND,
//   either express or implied.  See the License for the specific language governing
//   permissions and limitations under the License.
//------------------------------------------------------------------------
/* $Id: dpi_vpi.sv,v 1.6 2013/03/22 23:39:42 drich Exp $ */
package VPI_pkg;
  typedef chandle vpiHandle;
  // context is needed for calling VPI from DPI imported routines

  import "DPI-C" context
    VPI_handle_by_name = function vpiHandle vpi_handle_by_name(string name, vpiHandle scope);
  import "DPI-C" context
    VPI_iterate          = function vpiHandle vpi_iterate (int _type, vpiHandle _ref);
  import "DPI-C" context
    VPI_scan             = function vpiHandle vpi_scan (vpiHandle itr);
  import "DPI-C" context
    VPI_get_str          = function void _vpi_get_str (int prop, vpiHandle obj, output string name);
           function automatic string vpi_get_str(int prop, vpiHandle obj);
           string s;
           _vpi_get_str(prop,obj,s);
           return s;
    endfunction

  import "DPI-C" context
    VPI_handle           = function vpiHandle vpi_handle(int prop, vpiHandle obj);
  import "DPI-C" context
    VPI_get              = function int vpi_get(int prop, vpiHandle handle);
  import "DPI-C" context
    VPI_get_int_value    = function int vpi_get_int_value(vpiHandle handle);
  import "DPI-C" context
    VPI_put_int_value    = function void vpi_put_int_value(vpiHandle handle, int value);

/********************************* OBJECT TYPES ********************************/
           parameter vpiAlways          = 1;  /* always construct */
           parameter vpiAssignStmt      = 2;  /* quasi-continuous assignment */
           parameter vpiAssignment      = 3;  /* procedural assignment */
           parameter vpiBegin           = 4;  /* block statement */
           parameter vpiCase            = 5;  /* case statement */
           parameter vpiCaseItem        = 6;  /* case statement item */
           parameter vpiConstant        = 7;  /* numerical constant or string literal */
           parameter vpiContAssign      = 8;  /* continuous assignment */
           parameter vpiDeassign        = 9;  /* deassignment statement */
           parameter vpiDefParam        = 10;  /* defparam */
[ The complete set of constants appears in Annex K and M od the IEEE 1800-2012 LRM]
endpackage : VPI_pkg
```

DPI-C VPI Wrappers

```
#include "vpi_user.h"
#include "dpi_vpi.h"

vpiHandle VPI_handle_by_name(const char* name, vpiHandle scope) {
 vpiHandle handle;
 if(handle = vpi_handle_by_name((PLI_BYTE8*)name,scope))
   return handle;
 else {
    vpi_printf("VPI_handle_by_name: Can't find name %s\n", name);
    return 0;
 }
}
int32_t VPI_get_value(vpiHandle handle) {
 s_vpi_value value_p;
 if (handle) {
   value_p.format = vpiIntVal;
   vpi_get_value(handle,&value_p);
   return value_p.value.integer;
 }
 else
   vpi_printf("VPI_get_value: error null handle\n");
}
int32_t VPI_get(int32_t prop, vpiHandle obj) {
 if (obj) {
   return vpi_get(prop, obj);
 }
 else
   vpi_printf("VPI_get: error null handle\n");
}
vpiHandle VPI_iterate(int32_t type, vpiHandle ref) {
 return vpi_iterate(type,ref);
}
vpiHandle VPI_scan(vpiHandle itr){
 return vpi_scan(itr);
}
vpiHandle VPI_handle(int32_t type, vpiHandle ref) {
 return vpi_handle(type,ref);
}
void VPI_get_str(int32_t prop, vpiHandle obj, const char** name) {
 if (obj) {
   name[0] = vpi_get_str(prop, obj);
   return;
 }
 else
   vpi_printf("VPI_get_str: error null handle\n");
}
void VPI_put_value(vpiHandle handle, int value) {
 s_vpi_value value_p;
 if (handle) {
   value_p.format = vpiIntVal;
   value_p.value.integer = value;
   vpi_put_value(handle,&value_p,0,vpiNoDelay);
 }
 else
   vpi_printf("VPI_put_value: error null handle\n");
}
```